Microsoft®
Windows™95

CURSO IBM DE PROGRAMACION

17

EN Windows 95

IBM

PLANETA DeAGOSTINI

Microsoft®
Windows™95

CURSO IBM DE PROGRAMACION EN Windows 95

VOLUMEN 2 - UNIDAD 17

Director Gerente: José Mª Puig de la Bellacasa
Director Técnico: Francisco Valer Bañales
Director de Marketing: Javier Graupera Gassol
Director Editorial: Artur Klein Moreno
Directora de Software: Beatriz Cerrato Vidal

Dirección y coordinación: José Luis Becerril Gasco. De la Corporación IBM España, Licenciado en Ciencias Físicas, Director del Instituto de Inteligencia Artificial de IBM España.

Redactores y colaboradores: *José Ramón Dorronsoro Ibero.* Doctor en Ciencias Matemáticas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Vicente López Martínez.* Doctor en Ciencias Químicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Manuel Alfonseca Moreno.* Doctor Ingeniero de Telecomunicación y Licenciado en Informática, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Garrido Salas.* Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Martínez Rodríguez.* Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Santiago de Aréchaga Tarruell.* Ingeniero Superior de Informática, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Pedro Pascual Broncano.* Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Carlos Aguirre Maeso.* Licenciado en Ciencias Matemáticas, Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM).

Realización editorial: LH servicios informáticos, S.L.

Diseño gráfico: Esteoeste

© Multimedia Ediciones, S.A., Barcelona

ISBN Obra Completa: 84-88720-79-3
ISBN Fascículos: 84-88720-80-7
ISBN Volumen 2: 84-88720-82-3
Depósito Legal: B-33.798-97

Redacción y administración:
Aribau 198, 5ª planta - 08036 - Barcelona
Tel. (93) 306.78.90 - Fax (93) 306.78.97

Servicio de atención al cliente: Teléfono 902.101.324

Petición de números atrasados:
Teléfono 902.288.288
Apartado de correos nº 2.867, -08080- Barcelona

Servicio de consulta:
Teléfono 902.244.444 de lunes a viernes, de 10.00 a 14.30 y de 15.30 a 19.00 horas.
Fax (91) 661.67.08 durante las 24 horas del día.

Distribuye para España: MIDESA. Marco Ibérica de Ediciones, S.A., Carretera de Irún, km. 13,350, variante de Fuencarral - 28034 - Madrid

Venta en quioscos y librerías: Pida a su proveedor que le reserve sus ejemplares del curso. Adquiriéndolos todas las semanas en el mismo quiosco o librería facilitará la distribución y obtendrá un mejor servicio. El editor se reserva el derecho de modificar el precio de venta de los componentes de la colección en el transcurso de la misma, si las circunstancias del mercado así lo exigieran.

Composición del Curso: El Curso IBM de Programación se publica en forma de 52 unidades de aparición semanal y 52 CD-ROM. Las 52 unidades se encuadernan en 5 volúmenes. Con la unidad que completa cada volumen se ponen a la venta las tapas para su encuadernación.

Multimedia Ediciones garantiza la publicación de todos los elementos que componen esta obra.

© Editorial Planeta Argentina S.A.I.C.
Independencia, 1668 - Capital Federal
Teléfono atención al cliente: 482-0655 y 482-0657
Distribuye en capital: Huesca-Sanabria. En interior: D.G.P.
© Editorial Planeta Mexicana S.A. de C.V.
Avda. Insurgentes # 1162 - México D.F.
Teléfono de atención al cliente: 575-1348
© Editorial Planeta Colombiana S.A.
Carrera 68 A, nº 25-55 - Santafé de Bogotá
Teléfono de atención al cliente: 9800-15315

Imprime: Cayfosa. Carretera de Caldes, km. 3, Santa Perpetua de Mogoda (Barcelona).

INSTALACION/DESINSTALACION DEL SOFTWARE: Para la correcta instalación y desinstalación del CD-ROM que se entrega en esta unidad, véase el libreto que acompaña a cada CD-ROM.

Marcas comerciales registradas: En esta publicación son marcas registradas de sus respectivas compañías: Delphi Desktop, dBase, Turbo Pascal, Turbo C++, Turbo Assembler, Turbo Debugger, Borland Intrabuilder, Paradox, Visual dBase y BRIEF (Borland International, Inc.). Lotus, Lotus 123 y SmartIcons (Lotus Development Corporation, filial de IBM). Microsoft DOS, Microsoft Windows, Microsoft Windows 95, Excel, Visual Basic, Access y FoxPro (Microsoft Corporation). Netscape Navigator Gold (Netscape Communications Corporation). Java (Sun Microsystems, Inc.). IBM (International Business Machines).

CURSO DE PROGRAMACIÓN VISUAL: TURBO PASCAL (4)

MODULARIDAD

En esta unidad se aprenderán a utilizar "Funciones" y "Procedimientos", las estructuras que existen en Pascal para dividir un programa en bloques o módulos.

Con la sintaxis que hasta ahora se ha estudiado, pueden escribirse programas que realicen tareas muy complejas, pero resultan difíciles de escribir, leer o modificar. De entre todos los módulos del programa habrá uno, el principal, que será el que se ejecuta al correr la aplicación. Los demás entrarán en escena a medida que los invoque, directa o indirectamente, el principal. Se pretende que éste se pueda leer casi con la misma facilidad con la que se lee el pseudocódigo, que en él aparezcan llamadas a otros módulos que se entiendan con mucha facilidad, y que permita ver de un golpe qué tareas se han programado. En Pascal estos módulos se clasifican en Procedimientos (**Procedure**) y Funciones (**Function**).

INSTALACIÓN

El proceso de instalación de la unidad es muy sencillo y sigue los mismos pasos de entregas anteriores. Lo primero que hay que hacer es insertar el CD con el programa en el lector del equipo; después, abrir el menú Inicio y activar la opción Ejecutar; por último, suponiendo que el lector CD es la unidad D: del ordenador, hay que teclear:

```
D:\INSTALAR
```

Si el ordenador tiene activada la opción AutoRun de Windows 95, aparecerá la ventana de instalación sin necesidad de teclear ninguna orden. A partir de aquí aparecerá otra ventana en la que sólo habrá que elegir el disco donde ubicar Turbo Pascal y sus componentes. El programa de instalación copiará desde el CD los archivos necesarios, que se disponen en el directorio \TPW del disco elegido.

DIVIDIR PARA VENCER

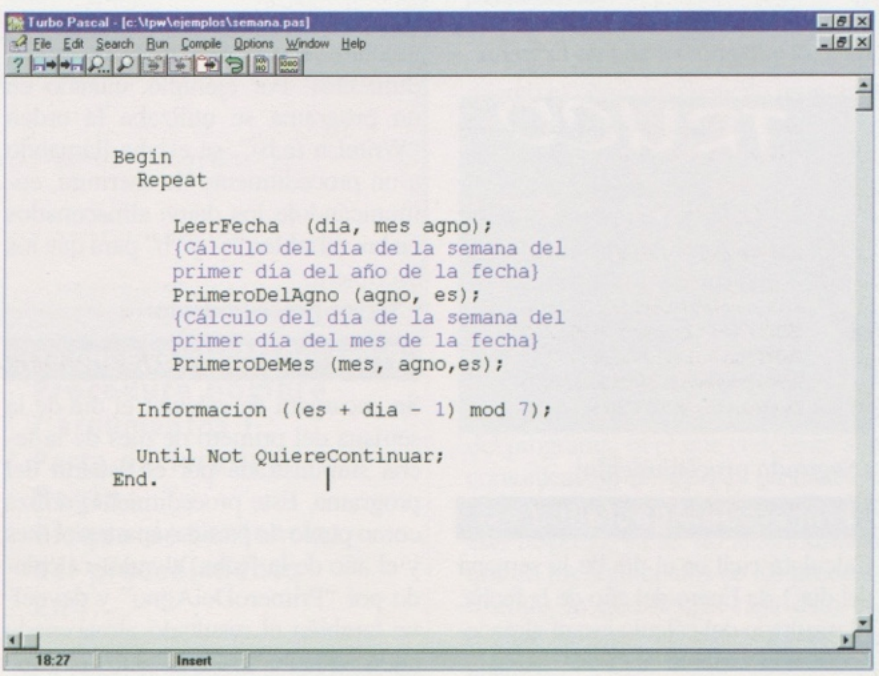
En programación es frecuente utilizar la expresión "dividir para vencer" al referirse a la estrategia de dividir un programa en módulos para facilitar su comprensión.

Esta estrategia resulta más útil para resolver un problema a medida que vaya aumentando el tamaño de la tarea que se haya de ejecutar para resolverlo. En el caso de la programación, se hace imprescindible cuando se tienen que escribir aplicaciones que están más allá de los ejercicios de aprendizaje.

En la programación, "Dividir para vencer" significa fragmentar un programa en partes más pequeñas, en bloques que actúan a su vez como pequeños programas. Cada uno de ellos tendrá sus datos de entrada y proporcionará como resultados, datos o acciones. A su vez, cada bloque habrá de ser suficientemente pequeño para que resulte efectivo el procedimiento que se diseñe, proporcionando la completa seguridad de que el "subprograma" ejecute bien lo que se espera de él.

Si se está seguro de que cada uno de los bloques es correcto, para construir bien el programa completo sólo quedará unirlos adecuadamente. Además, esta división en módulos permitirá reaprovechar de una vez para otra el código que se haya escrito.

Fig. 1. Bloque principal del programa Calendario.



Estos conceptos y su uso se aprenderán ilustrándolos con un caso concreto. Se trata de un programa que, a partir del día, mes y año actuales, contesta qué día de la semana es la fecha indicada. Al programa se le dará el nombre de "Calendario".

Uno de los objetivos de la modularidad es que el bloque principal del programa sea tan escueto y explicativo como el que puede verse en la figura 1, perteneciente al Programa "Calendario".

A la vista de este bloque principal es fácil entender lo que realiza el programa y el algoritmo elegido. En el programa se repite, mientras el usuario no indique que no quiere continuar, la llamada a cuatro procedimientos. Para entender mejor estas llamadas hay que observar las declaraciones de las variables que se utilizan en el bloque principal (figura 2).

Ahora se estudiará cuál es el objetivo de cada procedimiento:

• **Primer procedimiento:**

```
LeerFecha(dia,mes,agno);
```

Se invoca para captar la fecha para la que el usuario quiere conocer el día de la semana. Cuando se ejecute este procedimiento, la variable "dia" contendrá el entero correspondiente al día del mes. La variable "mes" un número que indique el mes del año (1 para Enero), y "agno" el año de la fecha.

TRUCO

Dado que no se puede utilizar la ñ en los identificadores, hay que recurrir a alguna alternativa para representar esta letra. En el caso de "año" pueden ser opciones "agno", "anyo", ...

• **Segundo procedimiento:**

```
PrimeroDelAgno(agno,es);
```

Calculará cuál es el día de la semana del día 1 de Enero del año de la fecha. El resultado del cálculo estará almacenado en la variable "es".

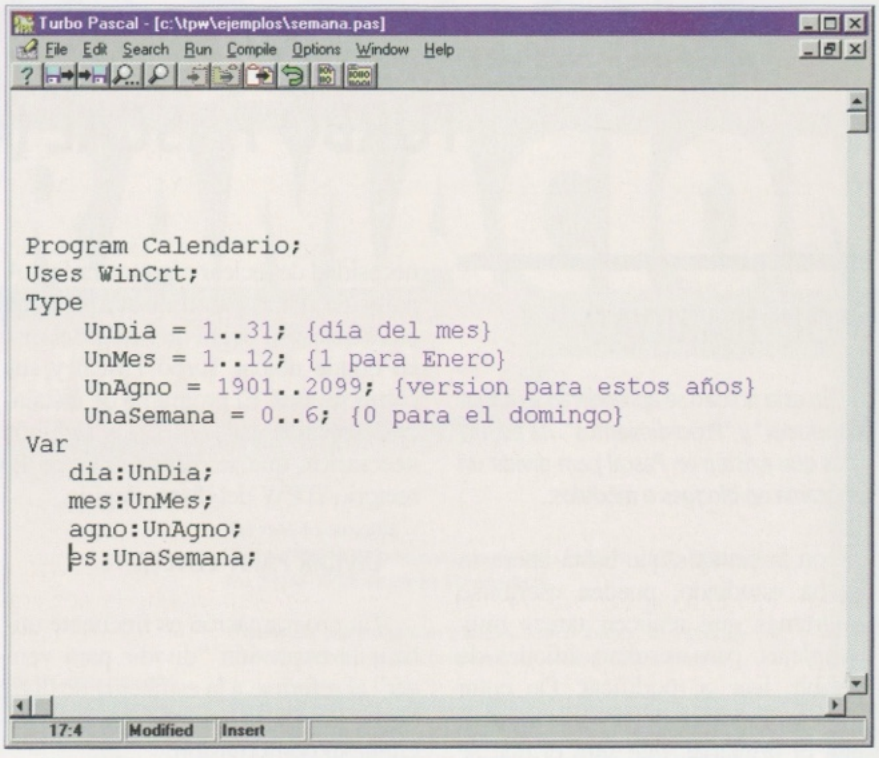


Fig. 2. Declaración de las variables que se utilizan en el bloque principal del programa Calendario.

La comunicación de datos entre el programa principal y los distintos módulos se realiza mediante los argumentos situados entre paréntesis y separados por comas. Todo esto resultará muy familiar a los seguidores del curso, porque ya han utilizado muchas veces varios de estos procedimientos. Por ejemplo, cuando en un programa se utilizaba la orden "WriteLn (a,b)", se estaba llamando a un procedimiento de escritura, comunicándole los datos almacenados en las variables "a" y "b" para que los escribiera.

• **Tercer procedimiento:**

```
PrimeroDeMes(mes,agno,es);
```

Se encargará de calcular el día de la semana del primero de mes de la fecha suministrada por el usuario del programa. Este procedimiento utiliza como punto de partida, aparte del mes y el año de la fecha, el valor calculado por "PrimeroDelAgno" y devuelve también el resultado almacenado en la variable "es".

• **Cuarto procedimiento:**

```
Informacion( (es + dia - 1) mod 7);
```

Es el encargado de comunicar el resultado al usuario. La operación que se realiza en el argumento es la que calcula el día de la semana de la fecha a partir del resultado para el primero del mes. Hay que explicar también que "QuiereContinuar" es una función que devolverá el valor **True** si el usuario ha manifestado su interés por seguir consultando fechas. A la vista del programa principal de "Calendario" está claro cuál es la sucesión de cálculos necesaria para dar la respuesta deseada. También es fácil concentrarse en cada uno de los bloques para encontrar un fallo si los resultados que proporciona la aplicación no son correctos. Por ejemplo, si el resultado que devuelve el bloque que calcula el día de la semana del primero del año es correcto, el usuario podrá concentrarse en los procedimientos que se llaman a continuación.

PROCEDIMIENTOS

La explicación de lo que realiza un procedimiento aparece en la parte

declarativa del programa. Allí donde se declara el significado de las constantes, o las características de las variables, se puede declarar el significado y características de los módulos de programa (procedimientos y funciones) que necesita conocer el compilador para interpretar el bloque principal del programa.

La declaración del procedimiento empieza con la palabra reservada Procedure (procedimiento), seguida del nombre elegido para el módulo.

A continuación, se sitúan entre paréntesis los nombres de los argumentos separados por comas. La instrucción finaliza, como todas, con el correspondiente símbolo ";".

Por ejemplo, la declaración del contenido del procedimiento "Informacion" puede ser la presentada en la figura 3.

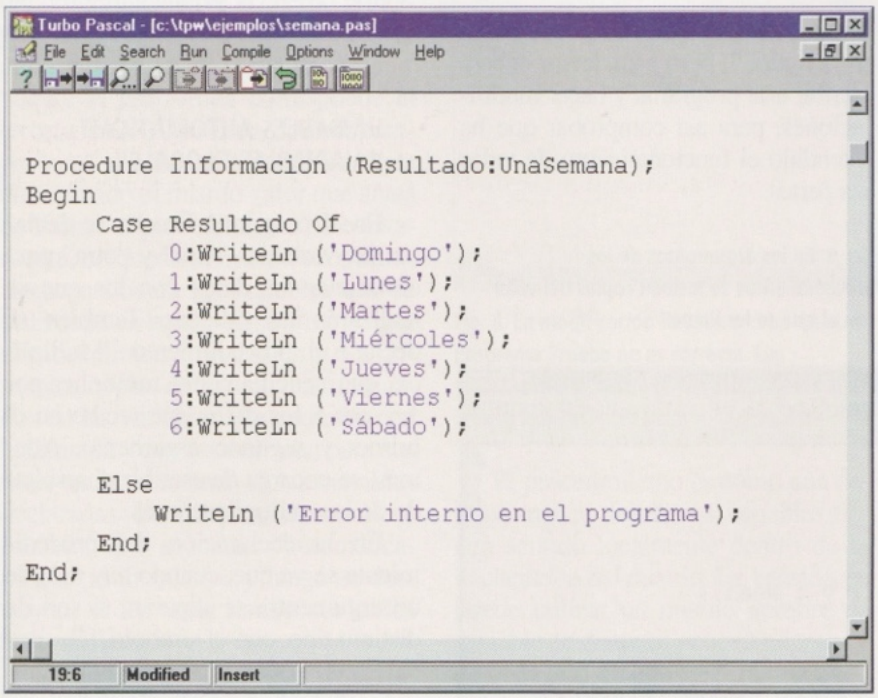


Fig. 3. Declaración de las tareas que ha de realizar el procedimiento "Informacion".

Observando la figura 3 puede comprobarse que el nombre del argumento no coincide con el utilizado en la llamada desde el bloque principal. Pues bien, no es necesario que coincidan y, si hay más de un argumento, se identifica cada uno por la posición que ocupa en la secuencia, no por el nombre. De hecho, un procedimiento es como un pequeño programa con sus propias variables, constantes e instrucciones ejecutables. Las variables del procedimiento se declaran en su correspondiente parte declarativa que sigue a la palabra reservada **Var**, pero las que aparecen en los argumentos tienen la declaración allí mismo. Ese es el motivo por el que en el argumento de la declaración de "Informacion" se añade ": UnaSemana" detrás del nombre de la variable "Resultado": para informar al compilador de que, en este caso, el argumento es un número entero en el rango "0..6" (figura 4).

Si se observa de nuevo la figura 3 puede sorprender que la declaración del procedimiento no acabe con un "End.", y que el último símbolo sea el ";".

```
Procedure nombre
( argumentos ) ;
Parte declarativa
Begin
Bloque de sentencias
del procedimiento
End;
```

Fig. 4. Estructura general de un procedimiento.

Pues bien, es bastante lógico. La declaración del procedimiento es una instrucción del Pascal y, como tal, acaba con el ";". El **End** empareja con el **Begin** con el que empieza la parte del procedimiento en la que se incluyen las instrucciones que debe realizar. Anteriormente, se llevan a cabo las declaraciones de las variables para que el compilador pueda interpretar correctamente esas instrucciones. En el caso de "Informacion", la única variable que se declara es la del argumento porque no se necesita ninguna más. Pero también se puede utilizar la modificación del procedimiento que puede ver en la figura 5, en la que se emplea la variable "mensa" para ampliar el contenido del mensaje.

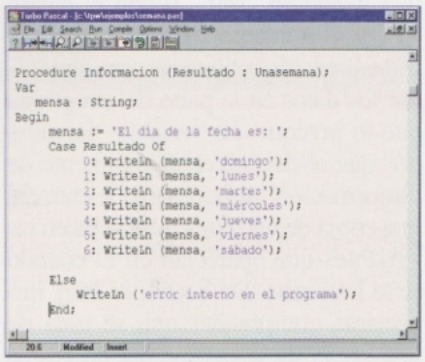


Fig. 5. Modificación de "Informacion" en la que se utiliza una variable interna del procedimiento.

Para muchos efectos, un procedimiento es como un pequeño programa Pascal en el que se realizan unas pocas tareas y que tiene sus propias variables. El procedimiento es independiente del contexto en el que está, es decir, el mismo (en este caso, bastante trivial) puede servir para otros programas, independientemente de las operaciones que se realicen en otros módulos.

El detalle fundamental que distingue sintácticamente el procedimiento del programa, es el que concierne a la comunicación de datos, constantes o variables a través del argumento. A este respecto, ya se vio anteriormente que en los argumentos de los procedimientos lo que define cada dato es, exclusivamente, el orden en el que aparece.

El procedimiento "Informacion" es llamado en el programa Calendario mediante

```
Informacion((es + dia - 1) mod 7);
```

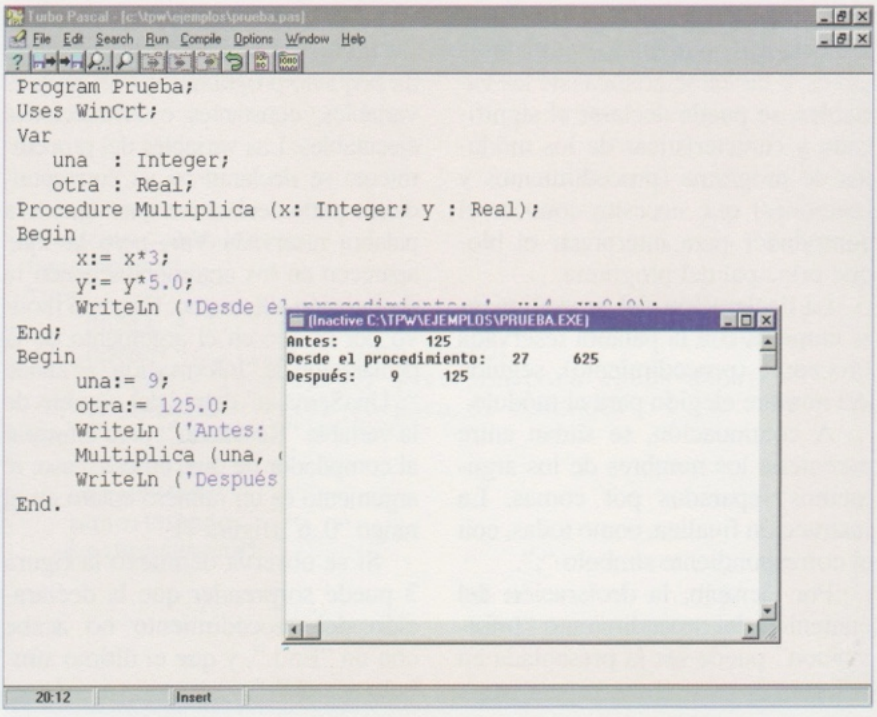
pero si se construye otro programa en el que sea necesario realizar la misma tarea, al procedimiento se le podría llamar con

```
Informacion(sem);
```

siempre que a "sem" se le haya declarado del tipo "unaSemana", o bien con una constante en el argumento

```
Informacion( 3 );
```

El procedimiento "Informacion" realizará las mismas tareas con el dato que reciba en "sem" o con el entero "3", y será independiente del nombre de las variables utilizadas para almacenar los datos en la parte del programa que lo invoque. La variable "Resultado" que se declara en el argumento de "Informacion" almacenará, de entrada, una copia de los datos que había en las variables que aparecían en él cuando se le invocaba. De hecho, es muy importante comprender que se trata de una copia de los datos. El programa Prueba (figura 6) servirá para comprenderlo mejor.



La salida del mismo puede verse en la figura 7, pero cada lector deberá escribir este programa y hacer modificaciones, para así comprobar que ha entendido el funcionamiento de todas sus partes.

Fig. 6. En los argumentos de los procedimientos se reciben copias del valor con el que se les llama.

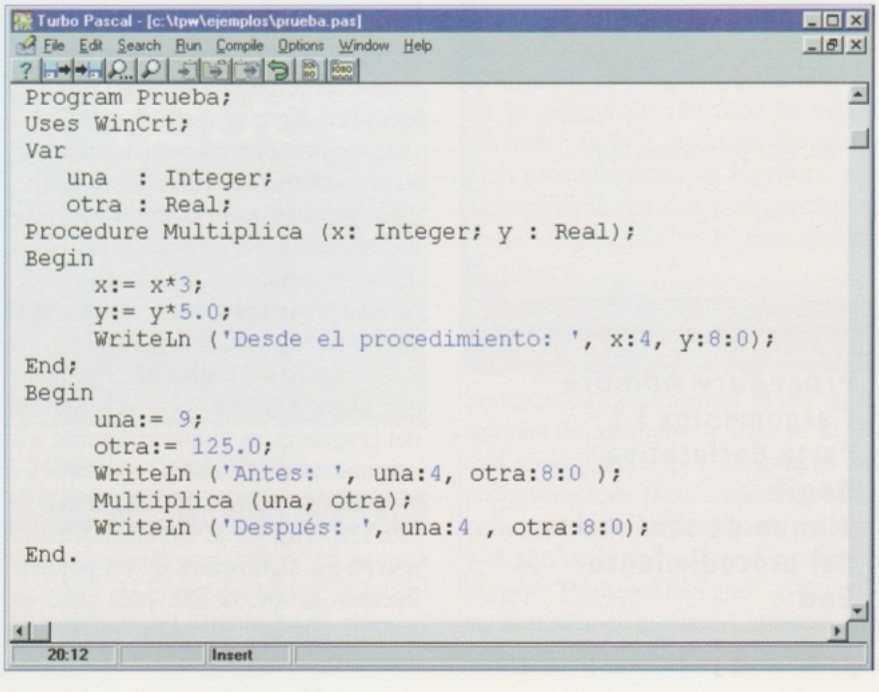


Fig. 7. Salida del programa Prueba.

VARIABLES AUTOMÁTICAS Y VARIABLES GLOBALES

En el programa Prueba se declaran las variables "una" y "otra" para almacenar los datos con los que se realizarán las pruebas. También se declara el procedimiento "Multiplica" que, sencillamente, multiplica por 3 y por 5 los datos que recibe en el primer y segundo argumento. Además, se encarga de escribir el resultado de las multiplicaciones.

En la declaración del procedimiento se ve que, cuando hay más de un argumento, se separan, si son de distinto tipo, con el símbolo ";".

En el bloque principal del programa se dan valores a "una" y "otra", escribiéndolos para dejar constancia de ellos antes de invocar al procedimiento. De hecho, la primera línea de la salida (figura 7) corresponde a esta orden de escritura. Seguidamente, se llama al procedimiento "Multiplica", por lo que el flujo del programa continuará con las instrucciones del procedimiento. En éste se realizan las multiplicaciones y se escribe el resultado.

La segunda línea de la salida corresponde a esta llamada a la función de escritura y se puede comprobar que la multiplicación ha sido correcta.

A las variables "x" e "y" han llegado correctamente las copias de los valores almacenados en "una" y "otra". Es importante mencionar que para todas las variables que se definen en un procedimiento se reserva lugar en la memoria del ordenador cuando se invoca y que desaparecen cuando acaba. Es decir, que cuando se devuelve el control desde "Multiplica" al programa principal, todas sus variables, en este caso sólo "x" e "y", desaparecen y no queda ningún rastro ni del valor que almacenaban ni del lugar que ocupaban en la memoria del ordenador.

Una vez finalizada la labor encomendada al procedimiento, el flujo del programa vuelve al cuerpo principal y se escriben de nuevo los valores almacenados en las variables "una" y "otra". A esta orden corresponde la tercera línea de la salida y puede comprobarse que en estas variables sigue almacenado el mismo valor que antes de llamar al procedimiento. Efectivamente, no quedó ni rastro de los datos almacenados en las variables que se utilizaban en aquél. Ya se advirtió que el procedimiento recibía en sus argumentos copias de los datos almacenados en las variables que aparecían en la llamada y que las utilizaba para sus fines exclusivamente. A las variables declaradas en un procedimiento, o función, que se crean "automáticamente" al invocarlos y luego se destruyen, se les suele llamar **variables automáticas.**

Todas las variables que se declaran en un procedimiento, incluidas las que se declaran en la especificación de los argumentos, son "locales" al procedimiento y sólo tienen sentido, para todos los efectos, dentro de aquél. También ocurre que el compilador sólo conoce su significado dentro de lo que constituye la declaración del procedimiento. Para comprobar esto de forma práctica, lo mejor es modificar el programa Prueba añadiéndole al final del cuerpo principal una nueva instrucción **WriteLn('Argumentos',x:4,y:8)**;.

Al compilar el programa modificado se recibirá un mensaje de error número 3: Unknown identifier (Identificador desconocido) señalando a la "x" (figura 8). El compilador no sabe a qué se refiere el programador con los nombres "x" e "y".

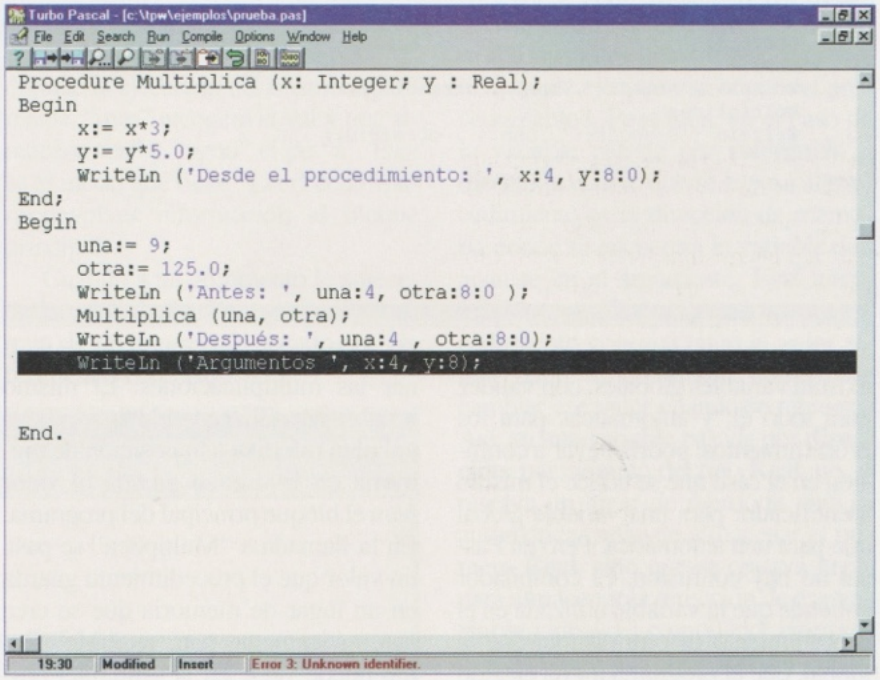


Fig. 8. La modificación llevada a cabo en el programa Prueba no es correcta. Las variables "x" e "y" no están declaradas en el bloque principal del mismo.

El procedimiento es como una caja hermética donde sus variables tienen sentido localmente dentro de la declaración del mismo. De entrada, se puede utilizar un mismo nombre de variable en distintos procedimientos y en cada uno de ellos tendrá una existencia distinta.

Se pueden también declarar procedimientos sin argumentos, pero en ese caso tienen poco sentido, ya que son únicamente fragmentos de programas puestos en un sitio distinto. Por ejemplo, en la modificación de Prueba que puede verse en la figura 9, el procedimiento "Multiplica" no añade nada nuevo. En ese caso la información llega al procedimiento a través de las variables "globales" declaradas al principio para todo el programa, incluidos los procedimientos subordinados.

Como puede verse en la salida del nuevo programa (figura 9), las modificaciones realizadas en el procedimiento de las variables "una" y "otra" perduran más allá del límite de aquél.

El resultado de la multiplicación se almacena en las mismas y únicas variables que se manejan en el programa.

A las variables que se declaran al principio del programa, antes de declarar ningún procedimiento, y que se pueden utilizar tanto en el bloque principal como en cualquiera de los procedimientos o funciones que se declaran a continuación, se les denomina **variables globales**.

TRUCO

Hay algunos casos en los que se puede llamar a procedimientos sin argumentos para intercambiar información con el usuario o con archivos externos.

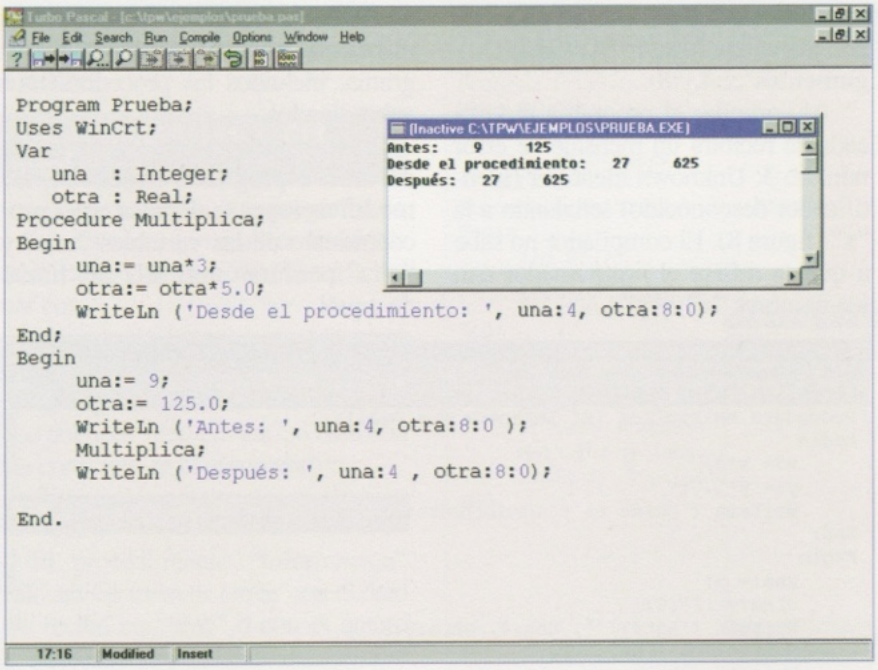


Fig. 9. La modularidad que se consigue con los procedimientos se pierde si no se utilizan argumentos.

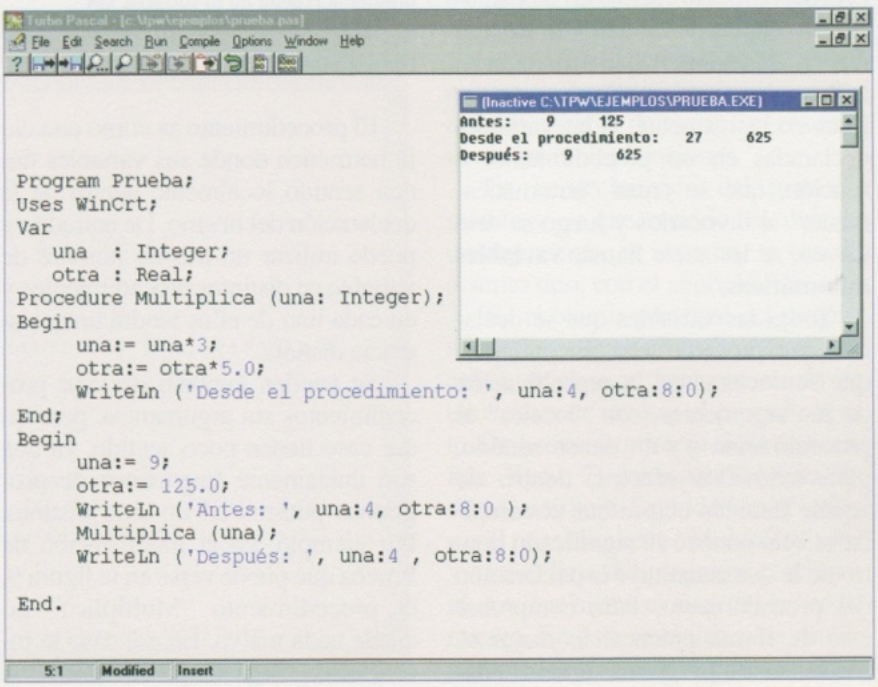
El hecho de que en un programa existan variables globales, con validez para todo él, y automáticas para los procedimientos, podría llevar a confusión en el caso que se utilice el mismo identificador para una variable global que para una automática. Pero en Pascal no hay confusión. El compilador entiende que la variable utilizada en el procedimiento es otra distinta que la global, que el programador ha elegido ese nombre para un lugar de memoria que se reservará en el procedimiento y que se liberará después de acabar.

Esto se entenderá mejor analizando el programa que puede verse en la figura 10. Se trata de una modificación del programa Prueba en el que se cambia el uso de las variables para hacerlo más enrevesado, pero permitiendo que se entienda mejor el ámbito de vigencia de las variables globales y automáticas, independientemente de las coincidencias de nombre.

La variable "una" que aparece en el procedimiento "Multiplica" es de tipo automático y no tiene nada que ver con la global que se define al principio del programa. Es una desgraciada casualidad que se llamen con el mismo nombre, pero el compilador no se confunde por esta coincidencia. Dentro del procedimiento "Multiplica" se crea una variable con el nombre de "una" que desaparece al terminar las multiplicaciones. El mismo nombre se utiliza en el bloque principal para referirse a la posición de memoria en la que se guarda el valor para el bloque principal del programa. En la llamada a "Multiplica" se pasa un valor que el procedimiento guarda en un lugar de memoria que se crea automáticamente para recibirlo. Como puede verse en la salida del programa (figura 10), la variable "otra" es global y cambia con la llamada al procedimiento.

En este punto el lector se estará preguntando si no ha llegado a un callejón sin salida. Si no existe una clara contradicción en lo dicho hasta ahora sobre los procedimientos. Por una parte, se insiste en que éstos han de escribirse para encapsular información y, además, que la información entre módulos se ha de trasvasar a través de los argumentos. Por otra parte, se ha visto que las variables automáticas que aparecen en un procedimiento, incluidos los argumentos, son locales y que no queda rastro de ellas durante la ejecución una vez que el procedimiento ha acabado su turno de actuación. Y por si todo lo dicho fuera poco, hay que añadir que el buen estilo de programación en Pascal exige que, salvo en contadas ocasiones, no se utilicen variables globales dentro de los procedimientos.

Fig. 10. Se pueden utilizar variables automáticas con el mismo nombre que las globales, pero esto no quiere decir que ambas coincidan o estén relacionadas.



Entonces, ¿para qué sirven los procedimientos si no hay forma de que devuelvan ninguna información a quien los llamó?

ARGUMENTOS PASADOS POR REFERENCIA Y ARGUMENTOS PASADOS POR CONTENIDO

Hasta ahora no se había tenido en cuenta una calificación que se puede dar a los argumentos para permitirles devolver información. Se va a explicar este detalle fundamental en la declaración del procedimiento "LeerFecha", que se utiliza en el programa Calendario para obtener la fecha de la que se quiere conocer el día de la semana. El procedimiento "LeerFecha", cuya declaración puede verse en la figura 11, no tendría sentido si no comunicara al bloque principal los datos que ha leído.

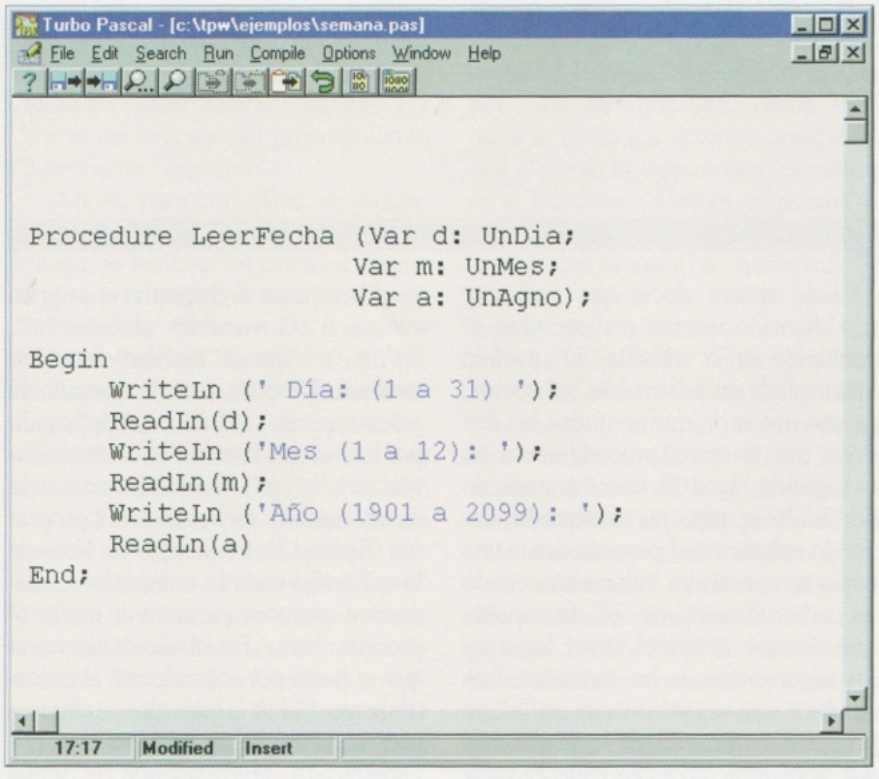


Fig. 11. Los argumentos antecedidos por el calificador Var se pueden modificar perdurablemente dentro del procedimiento.

En "LeerFecha" los argumentos "d", "m", y "a" aparecen en la declaración del procedimiento antecedidos por la palabra reservada **Var**. Con ello se especifica que las modificaciones que se realicen sobre el contenido de esa variable se han de trasladar a la variable que se utilizó en los argumentos de la llamada al procedimiento. Es decir, si se llamaba al procedimiento con

```
LeerFecha(dia,mes,agno);
```

al terminar de ejecutarse el procedimiento, la variable "dia" tendrá almacenado el valor que ha adquirido "d" durante la ejecución del mismo. Igualmente, "mes" recogerá el valor que almacene "m" y "agno" el de "a". Éste es el modo que tiene "LeerFecha" para devolver información al bloque principal.

Cuando a un argumento le antecede la calificación **Var** se dice que se trata de un argumento pasado por **referencia** o **dirección**.

Por el contrario, cualquiera de los vistos hasta ahora son argumentos pasados por contenido. El significado de estos nombres se explicará a continuación.

El compilador interpreta los programas para que el código al que dan lugar los procedimientos se cargue en la memoria del ordenador cuando se les invoca y utiliza copias de los datos que se le comunican en los argumentos. Esto sucede así siempre, independientemente del tipo de argumento. Entonces, ¿cómo es posible que el valor almacenado en una variable automática, que está declarada en los argumentos, perdure más allá de la existencia del procedimiento?. ¿Y cómo sucede esto si las variables automáticas desaparecen una vez que se ha terminado de ejecutar esa copia del procedimiento?. Pues bien, en el caso de la variable pasada por **referencia** o **dirección** el dato que se le pasa al procedimiento es la dirección de memoria donde se encuentra la variable que aparece en el argumento. Esta información es suficiente para que el procedimiento conozca tanto el valor almacenado en la variable como para cambiar ese valor cuando se requiera. Así, en una variable pasada por **dirección**, por ejemplo del tipo **Real**, no se reserva un lugar de memoria para la automática capaz de almacenar el número **Real**, sino que se reserva lugar para almacenar la dirección de esa **variable** global. El compilador interpretará que toda modificación del contenido de la variable automática se habrá de realizar allí donde se encuentra la original, tal como lo permite saber la dirección. Todos estos conceptos se comprenderán mejor cuando en la unidad siguiente se estudie el tipo de variable puntero que almacena direcciones de memoria.

PROGRAMA "LIO"

Ahora se va a utilizar un pequeño, pero complejo, programa para ayudar a asimilar estos conceptos. El programa, que puede verse en la figura 12, tiene un nombre adecuado y necesita cierta explicación, pero si se llega a comprender bien su funcionamiento, habrán quedado sólidamente establecidos los principios que gobiernan el intercambio de información entre los módulos de un programa. Estos pequeños programas que se utilizan pa-

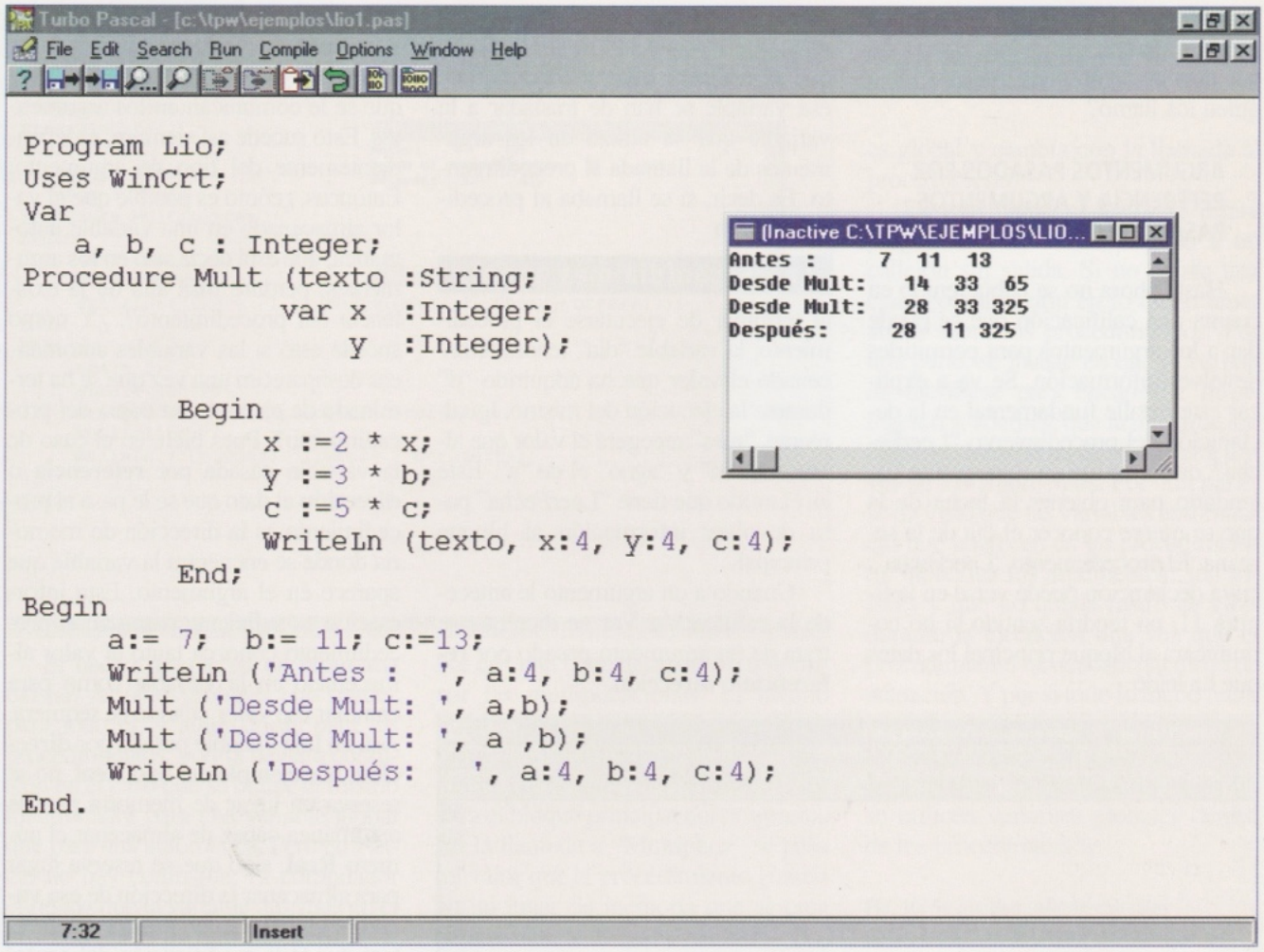


Fig. 12. Un buen puzzle para profundizar en el estudio de los ámbitos de validez de las variables.

ra ilustrar las ideas básicas son especialmente complejos, una especie de trabalenguas informáticos con los que se puede adquirir soltura en la programación.

El programa Lio utiliza el procedimiento "Mult" para ilustrar el ámbito de vigencia de las variables. Las de tipo global son "a", "b", y "c" y existen para todo el programa, incluido el procedimiento. "Mult" tiene como argumentos las variables "Texto", "x", e "y". El primero de los argumentos es una variable que almacenará datos del tipo **String** y se pasa por **contenido**. El segundo, "x", se pasa por **dirección**, tal como se indica al anteceder el nombre de la variable con la palabra reservada **Var**.

Esto quiere decir que cualquier modificación que se realice sobre el contenido de la variable "x" quedará almacenada en la variable "a" global, puesto que es la que se utiliza las dos veces para llamar al procedimiento, en el segundo lugar. El tercer argumento de "Mult" se pasa por **contenido**, llegando solamente al procedimiento una copia temporal del valor almacenado en cada momento en "b", la variable que siempre ocupa el tercer lugar en los argumentos de las llamadas. Los cambios que se introduzcan en "y" no perdurarán más allá de la realización del procedimiento y, por tanto, el valor almacenado en "b" no se alterará. Para completar las posibilidades, el procedimiento "Mult" también manipula el valor almacenado en la variable global "c" por lo que, todo cambio realizado sobre ella, se detectará desde cualquier lugar del programa.

Al empezar el programa se asignan valores a las variables globales "a", "b", y "c", que se escriben inmediatamente. Después entra en acción el procedimiento "Mult", multiplicando por los valores contenidos en las variables "x", "y", y "c", dejando constancia de los cambios en la salida del programa (figura 12). En la segunda línea de la salida del mismo, todas las cantidades son mayores que antes de llamar al procedimiento. En el momento en el que se llama por segunda vez al procedimiento "Mult", los valores almacenados en "a", "b", y "c" son, respectivamente, 14, 11 y 65. El de "b" no ha cambiado desde el principio, puesto que la multiplicación por 3 se realiza localmente en el procedimiento sobre la variable automática "y", que contiene inicialmente una copia del valor de "b". La segunda llamada al procedimiento "Mult" produce la tercera línea

de la salida y la cuarta la produce el bloque principal antes de finalizar. La salida del programa deja claro que el dato almacenado en la variable "b" nunca ha cambiado. Sin embargo, sí lo han hecho los de las variables "a" y "c" con las multiplicaciones realizadas en "Mult". En el caso de "a" porque se comunicaba al procedimiento por dirección y, en el caso de "c", porque se trataba de una variable global.

PROCEDIMIENTOS EN EL PROGRAMA CALENDARIO

Esta unidad comenzaba presentando el bloque principal del programa Calendario, que tiene como objetivo calcular el día de la semana de la fecha que proporcione el usuario.

Aparte del inicio del programa, con la declaración de las variables globales (figura 2) y el bloque principal (figura 1) se han visto también dos declaraciones posibles para el procedimiento "Informacion" (figuras 3 y 5) y la declaración del procedimiento "LeerFecha" (figura 11).

Ahora, para completar el programa, faltan por declarar los módulos en los que se realizan los cálculos. Según el bloque principal, se trata de los procedimientos "PrimeroDelAgno" y "PrimeroDeMes". También tendrá que declararse el indicador "QuiereContinuar" que, como se verá posteriormente, es una función que tendrá que devolver un valor Boolean, como bien se puede adivinar por el lugar donde aparece en el bloque principal.

• Procedimiento "PrimeroDelAgno"

Para comenzar, se va a intentar una posible declaración del procedimiento "PrimeroDelAgno" que calcula cuál es el día de la semana del 1 de Enero de un año que se le pasa como argumento (figura 13). El procedimiento utiliza el dato clave que es el día de la semana del 1 de Enero de 1901: martes.

Este dato se almacena en la constante "PrimeroDeEnero 1901" como el entero 2 (conviene recordar que el criterio escogido para las variables

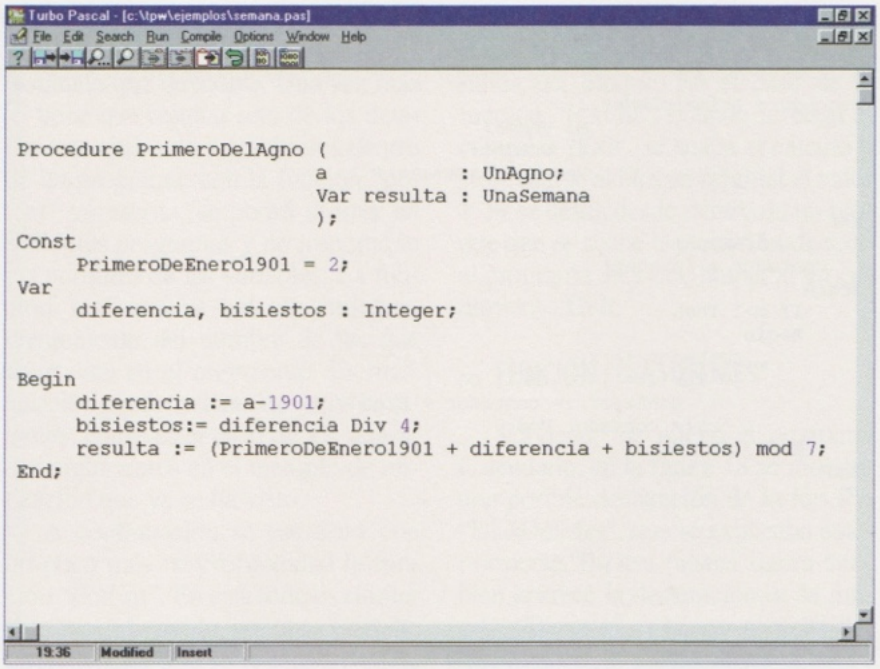


Fig. 13. Declaración del procedimiento "PrimeroDelAgno" que participa en el programa "Calendario".

del tipo "UnaSemana" es que el domingo corresponda al 0).

El otro punto clave para entender el procedimiento es darse cuenta de que, por cada año que pase desde 1901 hasta la fecha que interesa, avanza en uno el día de la semana que corresponde al 1 de Enero. Esto es así porque un año son 52 semanas más un día.

Pero no siempre hay que añadir un día más, puesto que en los años bisiestos esta regla no se cumple. Así pues, sumando la "diferencia" de años entre el de la fecha ("a") y 1901, y la existente entre el número de bisiestos transcurridos y el día de la semana del 1/1/1901, se obtendrá un número que, al dividirlo por 7, será el valor buscado para el 1 de Enero.

El cálculo del número de años bisiestos transcurridos, si se limita al rango entre 1901 y 2099, se puede calcular sencillamente contando por cuantos múltiplos de 4 se ha pasado. El resultado de este cálculo se devuelve en el segundo argumento. La variable "resulta" se ha pasado por dirección y, por tanto, en la variable "es" del bloque principal (figura 1) quedará constancia del valor calculado.

• Procedimiento "PrimeroDeMes"

Hasta ahora se han propuesto ejercicios que podían resolverse con programas pequeños. También se han utilizado programas sencillos para ilustrar las estructuras del lenguaje Pascal y con ellos se ha ejercitado el lector. Sin embargo ahora, guiados por la estrategia de dividir para vencer, se pueden escribir programas que resuelvan problemas complejos.

Cada procedimiento resolverá una tarea sencilla. A su vez, un procedimiento podrá llamar a otros y así será posible comprender un programa complejo y depurarlo fácilmente. Esto es lo que sucede en el procedimiento "PrimeroDeMes", que puede verse en la figura 14: el procedimiento llama a la función "DiasDelMes" que, como se verá después, llama a su vez a la función "Bisiesto".

El procedimiento "PrimeroDe Mes" calcula el día de la semana del día 1 del mes de la fecha que interesa, a partir del día de la semana del 1 de Enero del mismo año, dato éste que se encargó de calcular el procedimiento "PrimeroDelAgno". El argumento "m" recibe el entero correspondiente al mes (el 1 es Enero) y el argumento "a" recibe el año de la fecha.

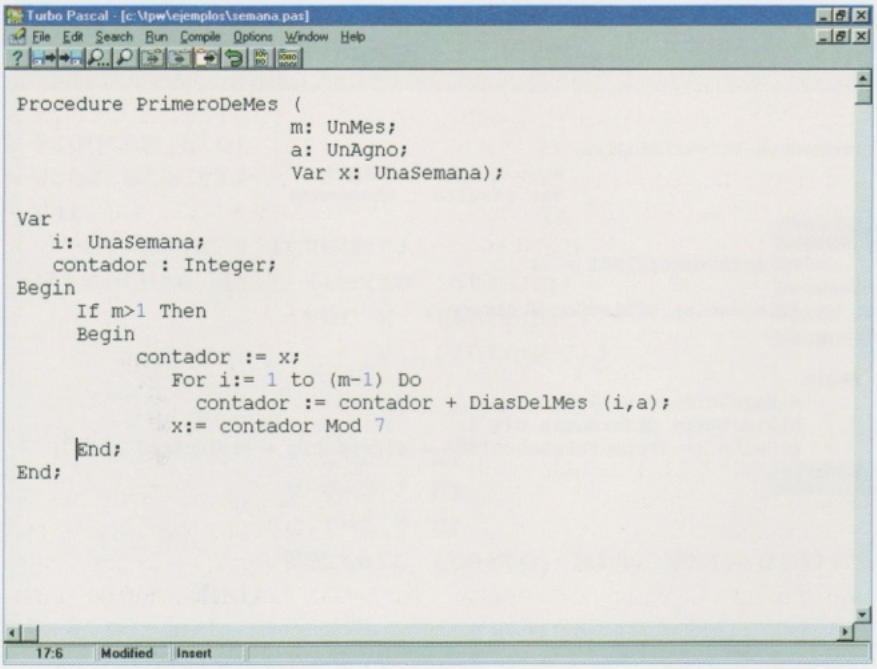


Fig. 14. Declaración del procedimiento "PrimeroDeMes" que participa en el programa "Calendario".

Ambos se pasan por contenido. Por último, el argumento "x", que se pasa por dirección, trae el dato del día de la semana del 1 de Enero (almacenado en la variable "es" del bloque principal, ver figura 1) y se encargará igualmente de transmitir al bloque principal el día de la semana que calcule para el 1 del mes almacenado en "m".

El algoritmo que utiliza la función "PrimeroDeMes" para calcular el día de la semana es muy sencillo. Se puede resumir tal como aparece en el recuadro:

Si es Enero no hay nada que calcular.
De lo contrario:
1.- Suma los días que hay en cada mes entre Enero y el deseado.
2.- Calcula el resto de dividir por 7 la suma de los días transcurridos.

FUNCIONES

Globalmente hay pocas diferencias entre procedimientos y funciones. La fundamental es que las segundas llevan asociado un valor a su nombre cuando se las invoca, de modo que la llamada a la función puede aparecer en medio de las expresiones del Pascal.

Una función devuelve un dato de un determinado tipo y su llamada puede aparecer en cualquier expresión o estructura donde esté permitido que aparezca ese tipo de dato. La expresión se evaluará normalmente, considerando que la función se sustituye por el dato que devuelve.

Ya se han utilizado funciones de las librerías estándar del Pascal en numerosas ocasiones. Por ejemplo, cuando se llamaba a la función "Ord" para que devolviera el ordinal de un carácter. Se utilizaba de tal modo que no importaba cómo efectuaba su trabajo, simplemente se tenía la certeza de que ahí donde estaba la llamada a la función quedaba sustituida por el número adecuado.

La sintaxis de la declaración de una función puede verse en la figura 15.

```
Function nombre
( argumentos ) : Tipodato ;
Parte declarativa
Begin
Bloque de sentencias de la función
End;
```

Fig. 15. Sintaxis de la declaración de una función.

Puede observarse que únicamente difiere de la del procedimiento en que al final de los argumentos se ha de incluir, tras dos puntos, el tipo de dato que devuelve. ¿Y cómo se especifica el valor que ha de devolver la función? Pues bien, entre las instrucciones de la función tiene que encontrarse una, al menos, en la que su nombre aparezca en la parte izquierda de una asignación. El valor último que se le ha asignado al nombre de la función es el dato que devuelve cuando finaliza.

En el procedimiento "PrimeroDeMes" aparece en la suma de días una llamada a la función "DiasDelMes" que se ha de encargar de calcular los días de cada mes a partir del número de orden del mes y del año (para considerar los meses de febrero de años bisiestos). Esta función es otro bloque más del programa "Calendario", sobre la que se volverá a tratar a lo largo de la unidad.

FUNCIÓN POTENT

Dejando por un momento el programa "Calendario", ahora se verá un

TRUCO

La variable "contador" es automática y declarada exclusivamente para el procedimiento. Es muy importante que esta variable se declare del tipo Integer y no del tipo "UnDia". La suma de los días puede fácilmente exceder 31, que es el rango máximo permitido para este tipo de variables. El compilador no avisará del error, pero se obtendrán en muchas ocasiones resultados absurdos.

ejemplo de declaración de función fuera de ese programa. El ejemplo está relacionado con la construcción de una función que quizás el lector piense que falta en las librerías de Turbo Pascal. Se trata de la función "PotEnt", que calcula el resultado de elevar un número a otro cuando ambos son enteros. Su declaración puede ser tal como aparece en la figura 16.

```
volumen := potent( x, 3)
+ potent(y,3) ;
```

Fig. 16. La función PotEnt

La función "PotEnt" toma como argumentos dos números enteros. El primero, "f", es la base y el segundo, "g", el exponente. Está diseñada para realizar el cálculo f^g y utiliza la definición más simple de la exponenciación. El resultado se calculará multiplicando la base tantas veces como dicte el exponente

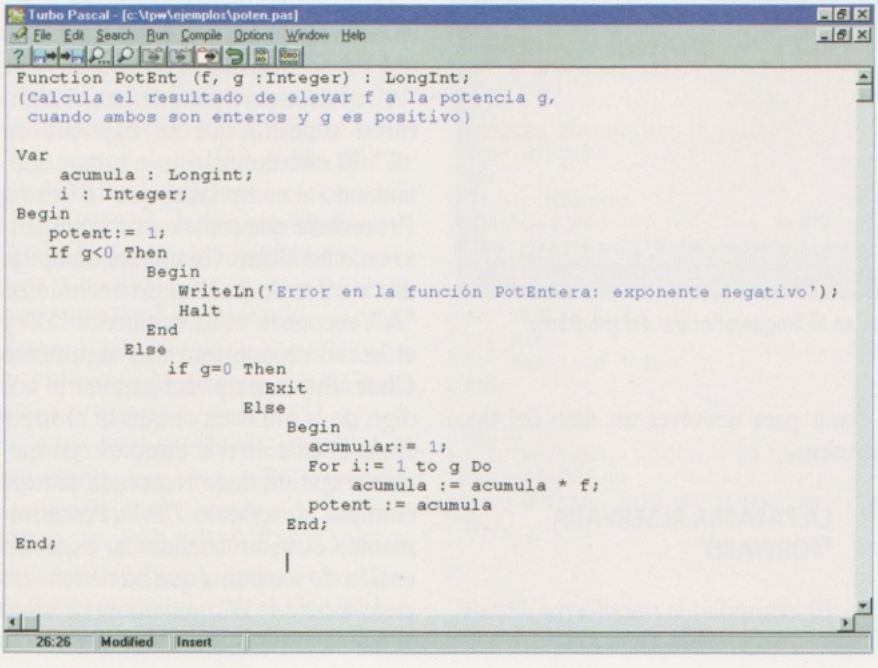
```
fg = f x f x f x ... x f
( g veces)
```

Antes de pasar a explicar ciertos detalles de la función "PotEnt" hay que describir su uso. Esta función se podrá llamar en medio de cualquier operación, aunque lo más frecuente es que se trate de una operación aritmética (figura 16).

La expresión se evaluará sustituyendo la llamada a la función por el resultado que devuelve. Una vez más se tiene que resaltar uno de los detalles más útiles de los módulos dentro de un programa: con la función "potent" ya escrita, se podrá utilizar en múltiples programas y no importarán los nombres de las variables. La función realizará el cálculo independientemente del nombre de las que aparezcan en el argumento. Es más, también puede tratarse de datos constantes, como es el caso de los segundos argumentos en el ejemplo de utilización que ya se ha visto.

A continuación se estudiará con un poco más de profundidad la función "PotEnt". En casi todo es similar a un procedimiento, aunque el detalle diferenciador está en la asignación de valores al nombre de la función. A lo largo de la declaración de aquélla se realizan dos asignaciones de valores al nombre "PotEnt". El valor que devuelve la función siempre es el que se ha asignado más recientemente a su nombre, antes de abandonarse el bloque. Hay que apuntar que la llamada al procedimiento **Exit** desde dentro de un bloque no da lugar a la finalización del programa, sino a la salida del bloque. En el caso de la función "PotEnt", cuando le llega el control a "Exit" se acaba el cálculo y se devuelve al bloque original el valor 1. Si se desea desde dentro de un bloque que se acabe la ejecución de todo el programa, hay que llamar al procedimiento **Halt**.

Fig. 17. Declaración de una función para el cálculo de potencias enteras de números enteros.



FUNCIÓN "DIASDELMES"

Volviendo de nuevo al programa Calendario, en la figura 18 se muestra una posible declaración de la función "DiasDelMes", que se explicaba anteriormente. En esa misma figura también aparece la declaración de la función "Bisiesto", a la que necesita llamar la función "DiasDelMes".

La función "DiasDelMes" es muy sencilla. Utiliza una estructura **Case** para elegir el número de días que devuelve. En el caso del mes 2 (Febrero) ha de consultar a la función "Bisiesto" si devuelve 28 ó 29.

La función "Bisiesto" devuelve un dato **Boolean**: el resultado de comparar con 0 el resto de dividir por 4 el año. Esta sencilla comprobación es válida exclusivamente para el rango de años que se mencionaron anteriormente.

La última función que se debe comprobar para tener todos los bloques que intervienen en el programa "Calendario" es "QuiereContinuar". Una posible declaración aparece en la figura 19. Al contrario que los procedimientos, las funciones pueden ser muy útiles aunque no tengan argumentos. Pueden devolver valor con el mismo nombre de la función y recibir los datos a través de una consulta al usuario o a un archivo.

Éste es el caso de la función "QuiereContinuar", que obtiene una respuesta del usuario y devuelve **True** en el caso de que esa respuesta empiece por la letra s, tanto mayúscula como minúscula. Para la decisión se tiene en cuenta la estructura interna de las variables **String** y se supone, arriesgadamente, que el usuario no va teclear ningún blanco antes del "si" o el "no".

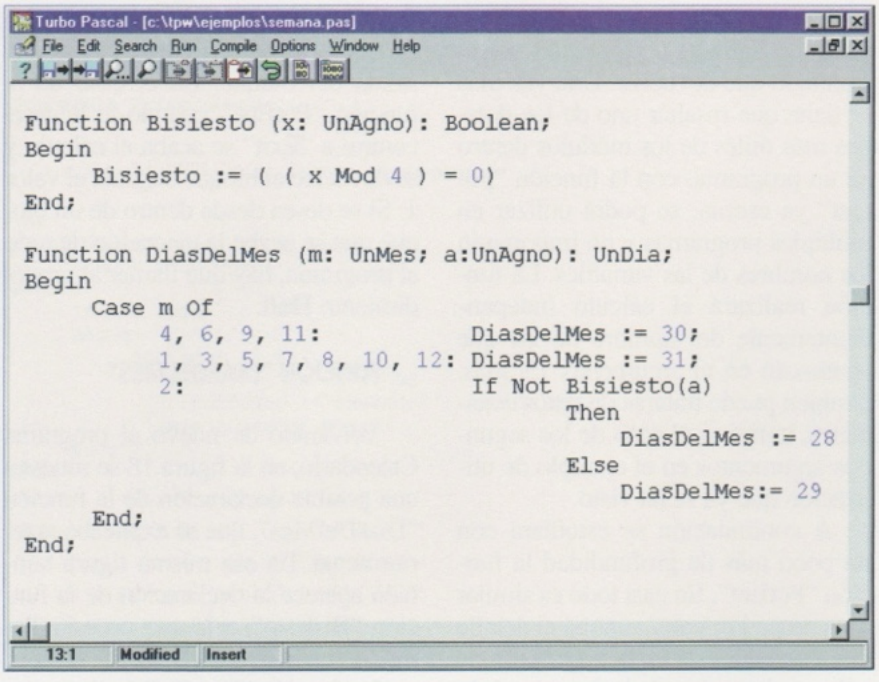


Fig. 18. Declaración de la función "DiasDelMes" que utiliza el programa Calendario y de la función "Bisiesto" que necesita llamar.

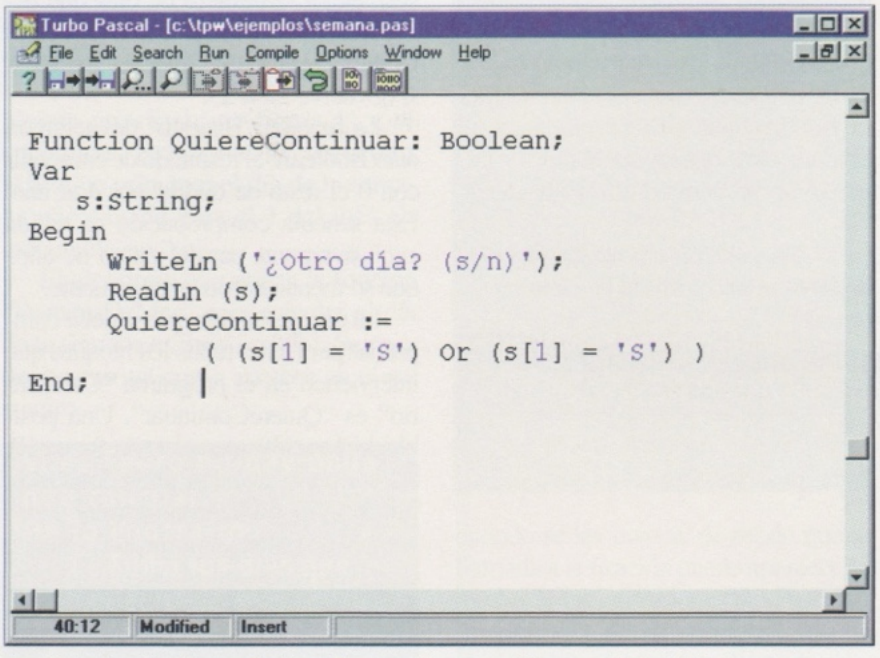


Fig. 19. La función "QuiereContinuar", que se llama en el bloque principal del programa Calendario.

Una restricción en la declaración de funciones es que no podrán devolver aquellos datos que no sea posible asignar directamente al nombre de la función.

Por ejemplo, una función no podrá estar declarada para devolver un **Array** pero, sin embargo, sí que se podrá declarar para devolver un dato del tipo **String**.

LA PALABRA RESERVADA "FORWARD"

El compilador transforma las órdenes Pascal a lenguaje máquina empezando por el principio del programa y acabando por el final. Por tanto, cualquier procedimiento o función ha de declararse antes de ser utilizado, con el fin de conseguir que el compilador sepa interpretar las instrucciones cuando lleguen al identificador correspondiente. Sin embargo, puede haber ocasiones en las que un algoritmo recursivo requiera que un módulo llame a otro y que, a su vez, éste llame al primero. En ese caso sería imposible definir los dos antes de referirlos. Para evitar este problema, Pascal proporciona la palabra reservada **FORWARD** (adelante) para seguir al nombre y argumentos de un procedimiento o función cuyo contenido se especificará en líneas posteriores. En esa línea sólo se declara que el identificador es el de un procedimiento o función con un determinado tipo de argumentos y se difiere para líneas posteriores la declaración de cuáles van a ser las líneas de código que deberá realizar. En el programa Listillo de la figura 20 puede verse cómo se utiliza la posibilidad de la declaración diferida (**FORWARD**) para crear un verdadero programa-puzzle. La salida del mismo es simplemente, tal como aparece en la esquina inferior derecha de la mencionada figura, la lista invertida de las letras mayúsculas.

En Listillo, el procedimiento "B" es necesario para definir el contenido del procedimiento "A" y, a su vez, "A" es necesario para definir el algoritmo siniestro que se explicita en "B". El circulo vicioso se rompe adelantando al compilador que "B" es un **Procedure** que toma como argumento un dato **Char**. Cuando el compilador analiza después el procedimiento "A", reconoce el identificador "B" y el hecho de que tome un argumento **Char**. Entonces podrá generar el código de la UCP en el que se redireccione el cálculo a la dirección de memoria que ya tiene reservada para la entrada al módulo "B". Posteriormente, cuando analice la especificación de los pasos que ha de realizar en el módulo "B", generará el código al que dará paso esa entrada.

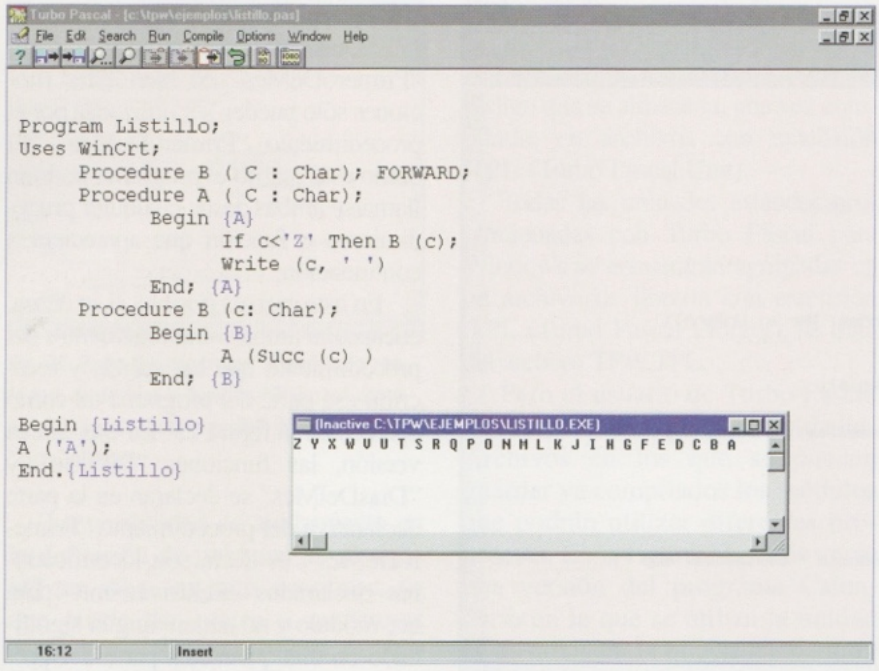


Fig. 20. Este programa utiliza la declaración diferida de un procedimiento anunciada por la palabra reservada Forward.

El programa Listillo, que no merece ningún comentario extra, se puede calificar como un claro ejemplo de mala programación. Al lector le resultará fácil comprender este programa si comprueba que contiene un algoritmo recursivo que suele utilizarse en muchos programas (por ejemplo el cálculo de un factorial). La recursividad en Pascal no se reduce solamente al ámbito de las funciones (figura 21), sino que también puede utilizarse con procedimientos. Es decir, los procedimientos pueden ser llamados de modo recursivo (figura 22).

A pesar de la impresión que le puede haber causado al lector este último programa, la recursividad se utiliza para mejorar la facilidad de escritura de los algoritmos y su legibilidad. Sin embargo, dan como resultado unos programas menos eficientes que otros que realizan la misma tarea pero sin usar la recursividad.

Muy frecuentemente se utiliza el algoritmo recursivo como primera aproximación a la resolución del problema. Posteriormente, un análisis más detallado puede llevar a otro algoritmo no recursivo que se ejecuta con menos gasto de recursos.

La primera versión recursiva, conceptualmente más sencilla, servirá para verificar los resultados del algoritmo más elaborado.

DIVISIÓN DE LOS PROGRAMAS

En esta unidad se comenzaba explicando la estrategia "Dividir para vencer". Ahora ya se sabe cómo aplicarla para resolver problemas usando programas de ordenador. Pero el lector puede preguntarse hasta qué punto es deseable aplicar la estrategia. La respuesta es que conviene dividir hasta que los módulos sean sencillos y fácilmente depurables. Su aplicación puede llevarse a cabo de dos modos muy distintos, como se verá a continuación. Al igual que para un programa se pueden definir módulos, también para un módulo se pueden escribir otros módulos. Es decir, declaraciones de procedimientos cuyo significado sólo conoce el módulo en el que se definen.

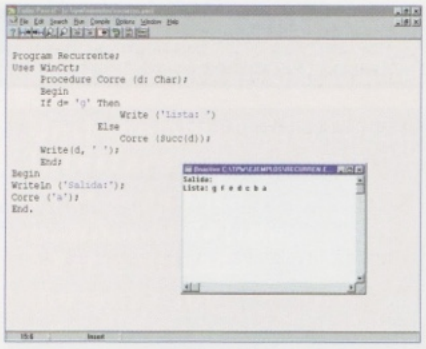
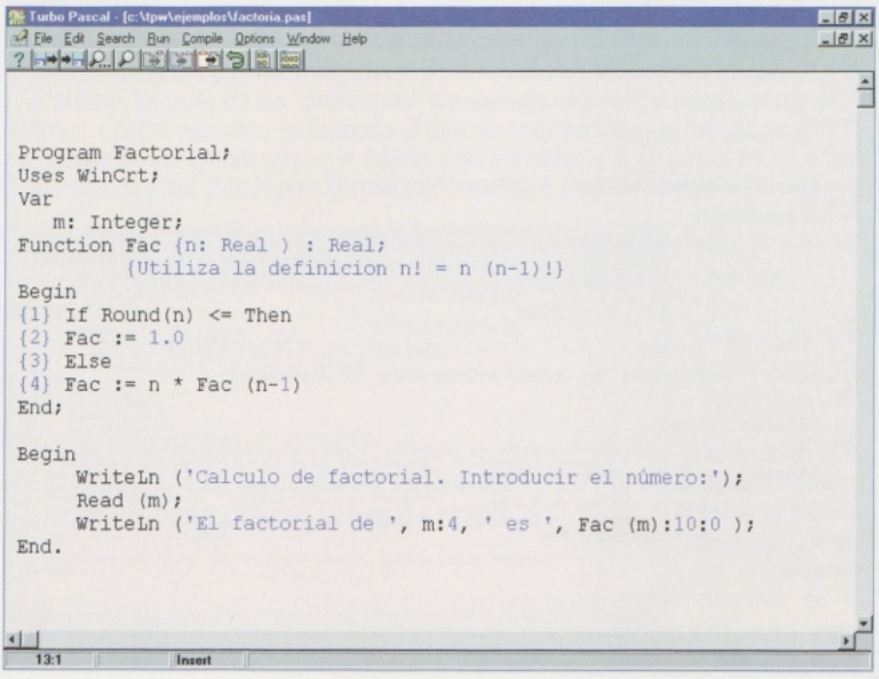


Fig. 22. La recursividad también puede utilizarse en los procedimientos.

Fig. 21. Este programa calcula recursivamente el factorial de un número.



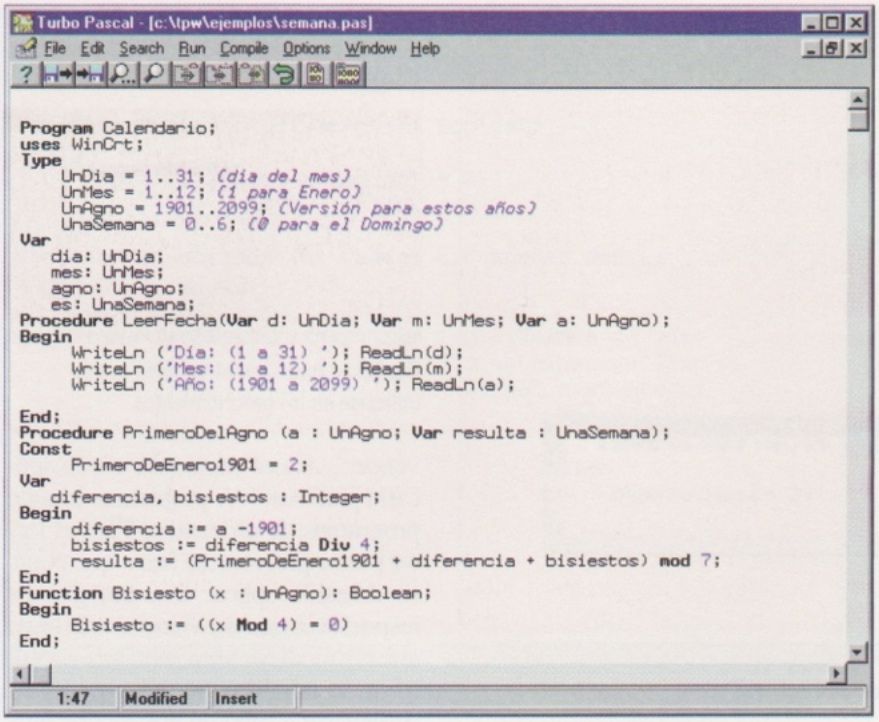
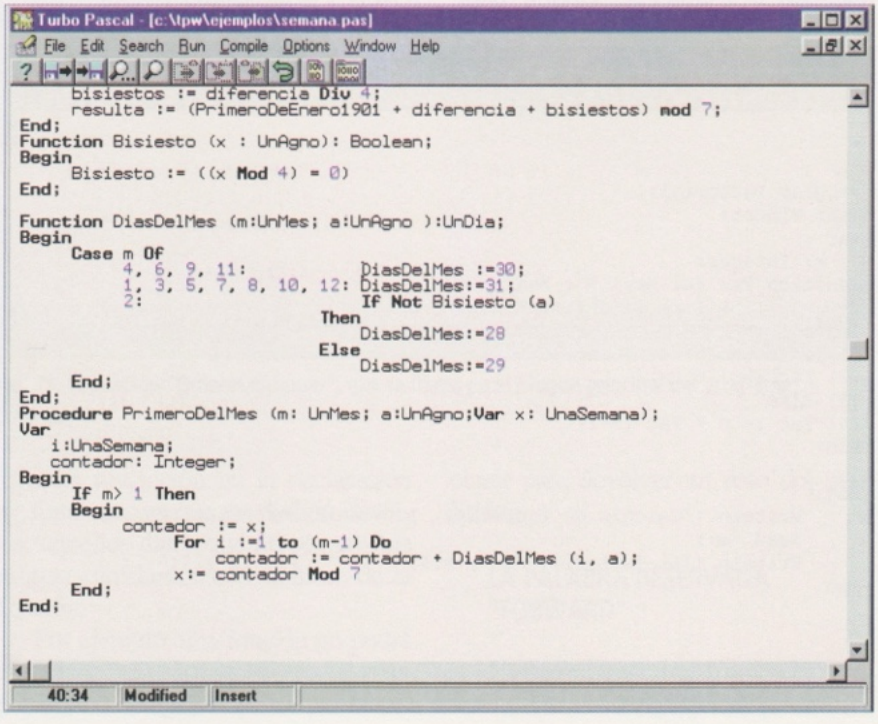


Fig. 23. Parte inicial del programa Calendario.

En la figura 23 puede verse el principio de la versión original del programa Calendario, en el que se definen todos los procedimientos y funciones, uno detrás de otro hasta el principal. Todos los procedimientos son accesibles desde el programa principal y para cada módulo también serán accesibles todos los otros que se hayan definido en líneas anteriores. En la figura 24 puede verse la zona del programa en la que se declaran las funciones "Bisiesto", "DiasDelMes" y el procedimiento "PrimeroDeMes". Si bien estas funciones sólo pueden ser utilizadas por el procedimiento "PrimeroDeMes", tal como está escrito el programa podrían llamarse ambas desde cualquier procedimiento o función que apareciera a continuación.

Fig. 24. Parte del programa "Calendario" en la que se declaran sucesivamente las funciones "Bisiesto", "DiasDelMes" y el procedimiento "PrimeroDelMes".



En este caso es posible, si se desea, encapsular ambas funciones dentro del procedimiento que las utiliza, y reescribir esa parte del programa tal como aparece en la figura 25. En esta nueva versión, las funciones "Bisiesto" y "DiasDelMes" se declaran en la parte declarativa del procedimiento "PrimeroDeMes", es decir, son identificadores declarados exclusivamente para ese módulo y no tienen ningún significado en cualquier otra parte del programa fuera de él.

Por ejemplo, si se llamara a la función "Bisiesto" desde el bloque principal del programa, el compilador avisaría del uso de un identificador que no conoce.

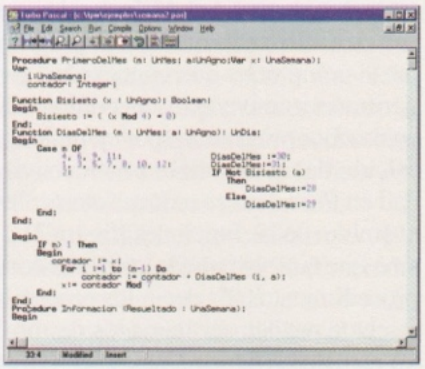


Fig. 25. Parte del programa "Calendario" en la que se declaran las funciones "Bisiesto" y "DiasDelMes" como módulos locales y sólo accesibles al procedimiento "PrimeroDeMes".

Dado que, a su vez, la función "Bisiesto" sólo la utiliza la función "Dias DelMes", se puede encapsular "Bisiesto" como un identificador que sólo sabe interpretar la función "DiasDel Mes". En esta tercera versión, esa parte del programa sería tal como puede verse en la figura 26. En este caso, ni siquiera en el procedimiento "Primero DeMes" se podría realizar una llamada directa a la función "Bisiesto".

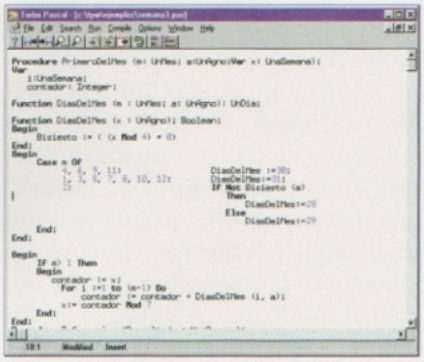


Fig. 26. Parte del programa "Calendario" en la que se declara la función "Bisiesto" como módulo local y sólo accesible a la función "DiasDelMes".

La posibilidad de anidamiento en la definición de módulos no se debe utilizar salvo en raras ocasiones, ya que puede dar origen a numerosas confusiones. Es mucho mejor declarar funciones o procedimientos, uno detrás de otro, de modo que tengan como ámbito de vigencia todo el programa. De hecho, es mejor aún seguir las recomendaciones del recuadro sobre prototipos.

UNIDADES EN TURBO PASCAL

Al dividir en módulos un programa se facilita el abordar problemas complejos, resolviendo uno a uno otros más sencillos.

Pero otro objetivo que ha de tener en cuenta el programador es escribir módulos reutilizables.

Cualquier programador puede construir sus propias librerías en las que puede almacenar módulos que reutilizará en proyectos sucesivos.

En Turbo Pascal las librerías se construyen a partir de las Unidades (ya se han utilizado algunas de ellas).

En todos los programas que se han visto hasta ahora se utilizaba la instrucción:

```
Uses WinCrt;
```

Con ella se indica que va a utilizarse la unidad denominada **WinCrt**, en la que queda englobado el código de los procedimientos de control de entrada y salida de datos a través de una ventana de Windows.

La palabra reservada **Uses** (utiliza) sirve para indicar que se utilizará la unidad. Estas unidades tienen un código que se almacena, una vez compilado, en archivos con extensión .TPU (Turbo Pascal Unit).

Todas las unidades estándar proporcionadas con Turbo Pascal para Windows se encuentran agrupadas en un archivo de librería con extensión .TPL (Turbo Pascal Library); se trata del archivo TPW.TPL.

Pero el usuario de Turbo Pascal puede crear sus propias unidades. Archivos en los que se pueden guardar ya compilados los módulos que podrán utilizar diferentes programas. En la figura 27 puede verse una versión del programa Calendario en la que se utiliza la unidad "Calenda", en la que se encuentran muchos de los módulos que necesita la aplicación.

Una unidad es un archivo en el que se escriben los módulos con la siguiente estructura:

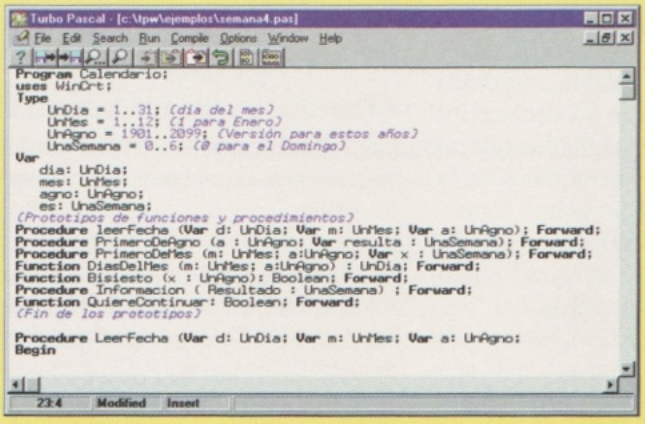
```
Unit nombre;
Interface
{ Parte pública }
Implementation
{ Parte privada }
Inicialización
End.
```

El archivo empieza con la palabra reservada **Unit** (Unidad), seguida del nombre de la misma. Se trata del mismo identificador que se utilizará en el programa principal tras la palabra reservada **Uses**. A continuación, la palabra reservada **Interface** antecede a las declaraciones de constantes, variables, funciones y procedimientos, que serán accesibles al programa que utiliza la unidad.

PROTOTIPOS

Cuando un programa tiene muchos módulos, puede resultar engorroso llevar la cuenta de cuáles se declaran antes y cuáles se declaran después, a fin de evitar que uno se utilice en una línea situada anteriormente a la declaración de ese módulo. La solución para este problema es muy sencilla y, además si el lector la utiliza frecuentemente, podrá irse acostumbrando a una práctica que es casi obligatoria en el lenguaje C. Se trata de la declaración de prototipos de funciones y procedimientos al principio del programa y que en Pascal utiliza la declaración diferida con la palabra reservada **Forward**.

En la figura puede verse una versión del programa Calendario en la que se declaran al principio todos los indicadores de funciones y procedimientos que se van a utilizar. Se trata de los "prototipos" que han de seguir cualquier llamada posterior a estos módulos, incluyendo el tipo de argumentos que han de recibir. Es irrelevante el nombre que se le asigne a las variables que aparecen en los argumentos de estos prototipos. El compilador se fijará exclusivamente en el número de ellos y en el tipo de dato de cada uno.



Después de esta declaración de "prototipos", se encontrarán las de los módulos, todos accesibles desde el bloque principal, siendo irrelevante el orden en el que se declaren.

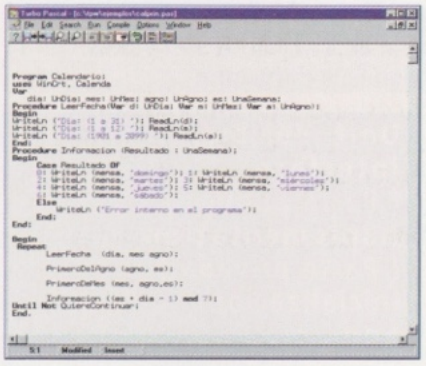


Fig. 27. Versión del programa Calendario en la que la mayoría de los módulos necesarios se encuentran en la unidad "Calenda".

Esta parte "pública" de la misma se acaba con la palabra reservada **Implementation**, que sirve para indicar que empieza la parte de la unidad en la que se declaran todas las funciones y procedimientos. Además, cuantas variables o constantes se declaren en esta parte sólo serán accesibles a la unidad y no al programa que la utilice. Las funciones y procedimientos se declaran completamente en esta parte y sólo aquellos módulos cuyo prototipo se incluía en la parte "pública" serán accesibles al programa que utiliza la unidad. Al final de la parte privada se puede incluir otra de inicialización, en la que se indican las instrucciones que se han de realizar antes empezar a utilizar cualquiera de los módulos que se incluyen en la unidad. La parte de inicialización se empieza con un **Begin** y acaba con el **End**. que siempre ha de estar presente al final del archivo de la unidad.

En las figuras 28 y 29 puede verse la unidad "Calenda" que completa la nueva versión del programa Calendario. Detrás de la palabra reservada **Interface** se incluyen las declaraciones de los tipos de variables que se utilizarán tanto en los módulos de la unidad como en el programa Calendario que se va a utilizar.

A continuación se incluyen los prototipos de los procedimientos "PrimeroDelAgno", "PrimeroDeMes" y la función "QuiereContinuar", que son los módulos a los que puede llamar Calendario.

Es una declaración similar a la que era necesaria para la declaración diferida de módulos con la opción **Forward**. Pero, en este caso, los indicadores que se utilicen para los argumentos han de ser los mismos que luego aparezcan cuando se declare el contenido del módulo. En la parte privada de la unidad, después de la palabra reservada **Implementation**, se incluye el código que constituye el cuerpo de las declaraciones de los procedimientos "PrimeroDelAgno", "Primero DeMes" y la función "QuiereContinuar". También se incluyen las declaraciones completas de las funciones "DiasDel Mes" y "Bisiesto".

La unidad acaba con la palabra reservada **End**. y no existe parte de inicialización.

Una vez que se ha escrito el contenido de la unidad en el archivo "calenda.pas", se podrá compilar con la opción **Compile|Compile** (o directamente con la combinación de teclas **Alt-F9**). Si no se ha producido ningún error sintáctico, se obtendrá el archivo "calenda.tpu" en el que se encontrará ya el código máquina que produce la unidad y que, posteriormente, será posible utilizar en el programa Calendario o en cualquier otro.

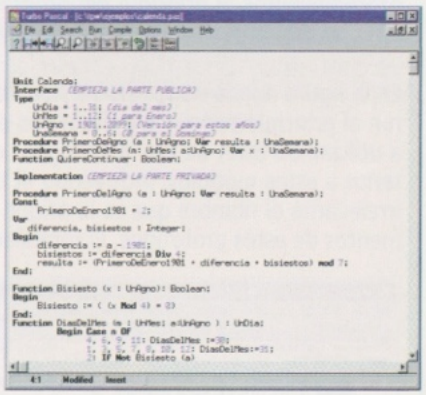


Fig. 28. Primera parte de la unidad Calenda necesaria para la nueva versión del programa Calendario.

Como ejercicio práctico se recomienda escribir la unidad "Calenda" en el archivo "calenda.pas" para producirla.

Después, podrá escribirse en otro archivo la nueva versión del programa Calendario, por ejemplo en el "calprin.pas".

Al elegir la opción **Compile | make** o la opción **Compile | Buid** se construirá el código ejecutable "calprin.exe" en el que se habrá incluido automáticamente el código necesario de la unidad "Calenda". También se "montará" la aplicación con la opción **Run | Run** y, además, se ejecutará el programa.

Las unidades se han de utilizar en Turbo Pascal para encapsular en archivos los módulos, ya compilados, que se escriban y que sean reutilizables por diferentes programas.

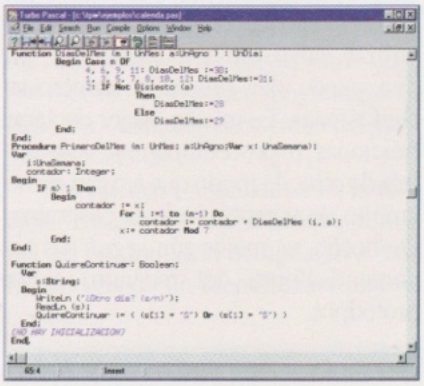


Fig. 29. Segunda parte de la unidad Calenda.

RESUMEN

Un programa se puede desmenuzar en módulos, cada uno de ellos diseñado para resolver uno de los problemas que se plantean al escribir la aplicación. Estos módulos, ya sean procedimientos o funciones, encapsulan los algoritmos y los datos de modo que no interfieran con otros presentes en otras partes del programa. Cada módulo será más fácil de escribir y depurar debido a que el desmenuzamiento se hará pensando en ese objetivo. Con los módulos se pueden también construir "unidades" que serán las librerías en donde el programador guarde todo aquel código que ha escrito y que piensa que podrá reutilizar en el futuro.

CUESTIONARIO AUTOEVALUATIVO

1 - La declaración de un procedimiento (**Procedure**) siempre debe aparecer:

- ❑ A - Antes del **Begin** con el que empieza la parte principal de un programa.
- ❑ B - Después del **Begin** con el que empieza la parte principal de un programa.
- ❑ C - Antes de la palabra reservada **Var** que antecede la declaración de las variables globales del programa.
- ❑ D - Después de la palabra reservada **Var** que antecede la declaración de las variables globales del programa.

2 - Se llama variables **automáticas** a las que:

- ❑ A - Se declaran automáticamente por ser un tipo de dato estándar del Pascal.
- ❑ B - Se declaran al principio de un programa y son automáticamente visibles por todos los módulos.
- ❑ C - Se declaran en los procedimientos y funciones y se crean automáticamente en la memoria del ordenador al invocar el módulo.
- ❑ D - Se crean automáticamente por aparecer en el argumento de un procedimiento.

3 - Un argumento antecedido por la palabra reservada **Var** en la declaración de un procedimiento o función:

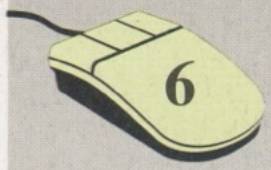
- ❑ A - Se comunica para efectos de contenido.
- ❑ B - Podrá utilizarse para devolver valores al programa que llama al módulo.
- ❑ C - No podrá utilizarse para devolver valores al programa que llama al módulo.
- ❑ D - Se comunica para efectos variables.

4 - Si el procedimiento "Copia" se ha declarado: **Procedure** Copia (a,b: **Integer**); **Begin** b:= a; **End**; después de llamar a "Copia" con: **Copia(5,x)** ¿qué valor se tendrá almacenado en la variable "x" del tipo **Integer**?

- ❑ A - 5
- ❑ B - El mismo que tenía la variable "b" la última vez que se llamó a "Copia".
- ❑ C - El mismo que tenía la variable "x" antes de llamar a "Copia".
- ❑ D - El mismo que tenía la variable "a" la última vez que se llamó a "Copia".

5 - Si la función "Sum" se ha declarado: **Function** Sum (a,b: **Integer**): **Real** ; ¿qué ocurre con la instrucción: res:= Suma(x,3) * Sum(4, 5);?

- ❑ A - Se calculará utilizando aritmética del tipo **Integer**.
- ❑ B - Se calculará utilizando aritmética del tipo **Real**.
- ❑ C - Es correcta sintácticamente sólo si la variable "res" se ha declarado del tipo **Integer**.
- ❑ D - Es incorrecta sintácticamente.



6 - Si se ha definido el tipo de dato: **Type** ristra = **Array**[1..80] **Of** Char; ¿será correcta la declaración de la función: **Function** Suma (a: ristra) : ristra;?

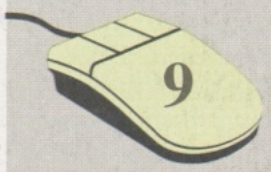
- ❑ A - Sí lo será sintácticamente.
- ❑ B - Sí, solamente si en la declaración de la función se le asigna al nombre "ristra" un dato del tipo **String**.
- ❑ C - No.
- ❑ D - Sí, siempre que no se hagan llamadas recursivas.

7 - Si al principio de un programa se declara una variable global y en un procedimiento se utiliza el mismo nombre para declarar una variable, ¿qué sucederá?

- ❑ A - El compilador detectará un error sintáctico.
- ❑ B - Se detectará un error durante la ejecución del programa.
- ❑ C - El compilador entenderá que se trata de dos variables distintas para todos los efectos.
- ❑ D - El compilador entenderá que se trata de una misma variable para todos los efectos.

8 - Una función que se declara en la parte reservada para las declaraciones de variables de un procedimiento:

- ❑ A - Sólo estará definida para llamadas que se realicen dentro de ese procedimiento.
- ❑ B - Dará lugar a un error en compilación.
- ❑ C - Dará lugar a un error en ejecución.
- ❑ D - Estará definida para cualquier procedimiento o función del programa.



9 - Detrás de la palabra reservada **Interface**, ¿qué se incluye en una unidad?

- ❑ A - Las declaraciones de funciones y procedimientos que controlan la entrada y salida de datos.
- ❑ B - Las declaraciones de variables y constantes globales que aparecen en el programa que llama a la unidad.
- ❑ C - Las declaraciones de variables, constantes, funciones y procedimientos que se reservan exclusivamente para el uso en la unidad.
- ❑ D - Las declaraciones de variables, constantes, funciones y procedimientos que podrá conocer el programa que utilice la unidad.

10-El archivo que contiene el código de una unidad siempre acabará con:

- ❑ A - **End**.
- ❑ B - **End**;
- ❑ C - Una parte de inicialización.
- ❑ D - La parte denominada **Interface**.

Respuestas: 1 A - 2 C - 3 B - 4 C - 5 B - 6 C - 7 C - 8 A - 9 D - 10 A